VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Ovládání počítače pohybem očí

Operating PCs movement eyes

2009 Aleš Wojnar

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. 5. 2009

.....

podpis

Poděkování

Chtěl bych poděkovat doc. Ing. Lačezaru Ličevovi, CSc. za vedení, návrhy a připomínky, které pomohly ke vzniku této práce.

Abstrakt

Tato práce se zabývá realizací komplexního programu, který dokáže za pomoci video kamery snímat lidské oko a vyhodnocovat tak jeho pohyb a stav. Program analyzuje pohyby oka pomocí metody prahování snímků, v nichž poté vyhledává polygony, u kterých vyhodnocuje jejich proporce a vychýlení středových os od středu vztažné roviny. Na základě takto získaných dat a po provedení kalibrace celého systému program dokáže vyhodnotit, jakou rychlostí se oko pohybuje nebo jestli je oko zavřené, či otevřené. Tyto faktory poté umožní uživateli programu ovládat PC myš a klávesnici pouze svým pohybem oka.

Klíčová slova

Java Media Framework, určování pohybu oka, prahování, detekce polygonů

Abstract

The thesis is focusing on the possible realization of complex program, which able to scan the human eye by the video camera and therefore analyze its movements and status. The program analyzes movements of the eye by the trashholding method, fatherly, the program is searching for the polygons which evaluate its proportions and deflection of the central axis from the centre of the basic plane. On the bases of collected data, and calibration of all system, program is able to evaluate either the speed of the eye movement or the eye state - closed or opened. These factors therefore allow the user to operate the computer mouse and the keyboard only by the eye movements.

Keywords

Java Media Framework, eye state determination, tresholding, polygon detection

Seznam zkratek a použitých symbolů

BFS - Breadth first search, proheldávání do šířky
CMOS - Complementary Metal–Oxide–Semiconductor, doplňující se kov-oxid-polovodič
FPS - frames per second, snímků za sekundu
GUI - Graphical User Interface, grafické uživatelské rozhraní
HID - Human Interface Device
JMF - Java Media Framework
PC - personal komputer, osobní počítač
RGB - barva, která je kódovaná pomocí tří barevných složek (červené, zelené, modré)
VOG - Videookulografie

Obsah

1 Úvod

1.1 Motivace

Nastává doba, kdy tradiční periferní zařízení počítače, které mají za úkol usnadnit uživateli jeho obsluhu, využívají nejnovějších poznatků a nejmodernějších technologií v daném vědním oboru. Klasickou kuličku, snímající polohu myši, nahradil laser s vysokorychlostním senzorem pro snímání pohybu. Existují také alternativní řešení, kdy myš byla nahrazena dotykovou podložkou nebo rovnou celým dotykovým displejem monitoru. Dnešní klávesnice už taky nemusí být tvořeny jenom plastovými tlačítky. Můžou být celé z tenké gumy, která umožní klávesnici srolovat jako list papíru, nebo můžou být zastoupeny laserovým projektorem, který „tlačítka“ pouze na projekční plochu (např. stůl) promítá. Uživatel přikládá prsty na výsledný promítaný obraz a systém tyto pohyby vyhodnocuje. Tato řešení určitým způsobem vybočují ze zavedené koncepce ovládání PC, nicméně je zde stále zapotřebí lidské ruky a prstů. Tato podmínka proto způsobuje, že počítač nemůže ovládat úplně každý člověk, kterému to nedovoluje míra postižení jeho svalového ústrojí a zhoršená motorika končetin. Pro tuto skupinu lidí je zapotřebí přijít se zcela nekoncepčním řešením problematiky ovládání PC. Jednou s možností je právě analýza pohybu oka, jelikož tento lidský orgán bývá zcela funkční i při těžkém stupni poškození svalového ústrojí.

1.2 Cíl práce

Cílem této práce je navrhnout systém, který by za pomoci běžně dostupných hardwarových a softwarových prostředků umožnil uživateli ovládat PC pohybem oka. To znamená, že je nutno vybrat vhodné zařízení na snímání obrazu oka, vybrat vhodné algoritmy pro analýzu pohybu oka, vybrat vhodnou technologii, s jejíž pomocí se navrhne a implementuje program, který na základě analýzy pohybu oka převezme funkce PC myši a klávesnice.

1.3 Struktura textu

V následující kapitole *Videookulografie a fyziologie lidského oka (2)* je popsán princip fungování lidského oka a definice videookulografie. V kapitole *Detekce polygonů v obraze (3)* jsou popsány vhodné algoritmy pro detekci polygonů a je zde proveden návrh implementace vybraného algoritmu. Kapitola *Existující řešení (4)* pojednává o komerčním systému uvedeném na trh pod názvem *I4Control*. V kapitole *Návrh systému EyeControl (5)* je zaznamenán postup, který by se měl dodržet při realizaci systému. V kapitole *Realizace systému EyeControl (6)* jsou popsány třídy a funkce systému. Kapitola *Testování systému (7)* se zabývá správnou funkcí systému. Poslední kapitola *Závěr (8)* shrnuje a hodnotí celý projekt.

2 Videookulografie a fyziologie lidského oka

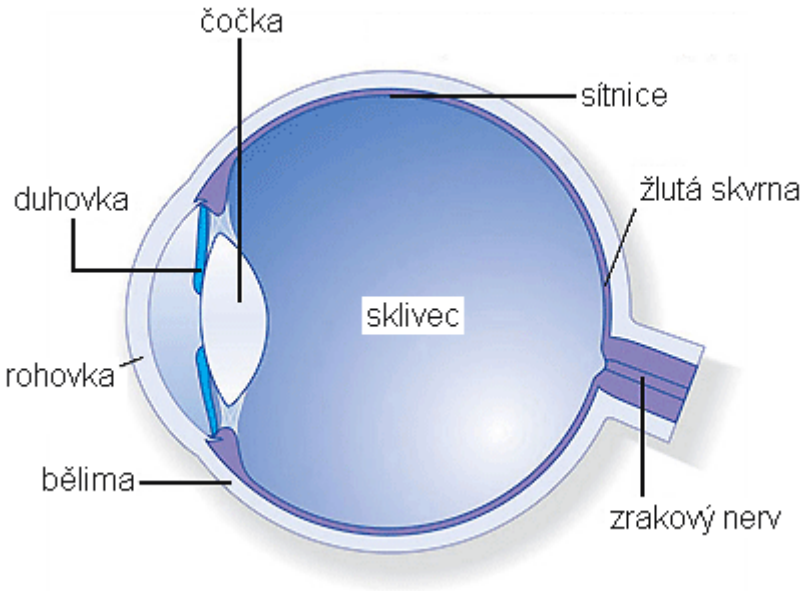
V této kapitole je popsána základní fyziologie a anatomie lidského oka, které v této práci vykonává funkci hlavního řídícího nástroje.

2.1 Definice Videookulografie

Videookulografie (VOG) je bezkontaktní postup, při němž je oko sledováno libovolným optickým snímacím systémem a tento záznam je později, případně dle použitých zařízení i v reálném čase vyhodnocován z hlediska sledovaných veličin. Těmi mohou být libovolné vizuálně patrné kvality, resp. jejich změna v čase. Obvykle to bývají amplituda či trvání fixací, sakád, tremoru, průměr zornice, mrkání a podobně. Princip vlastní funkce je poměrně jednoduchý, systém musí ve snímaném obrazu nalézt oko a přesně odečíst jeho polohu a další požadované informace. *Gaze-tracking* systém (sledující směr pohledu) navíc ze shromážděných geometrických dat aproximuje bod v předem určené vztažné soustavě, do něhož se promítá vektor pohledu sledovaného člověka.[8]

2.2 Fyziologie lidského oka

Lidské oko je velmi důmyslná optická soustava. Umožňuje vytvářet obraz na sítnici, z níž se informace o obrazu přenášejí slabými elektrickými proudy do mozku. Z hlediska paprskové optiky můžeme oko považovat za soustavu se spojnou čočkou, jejíž ohniskovou vzdálenost můžeme měnit.



Obrázek č. 1: Anatomie oka [5]

V oku světlo prochází nejprve průhlednou rohovkou, která má značnou optickou mohutnost (asi 40 D) [4]. Průměr očního otvoru se reguluje duhovkou umístěnou mezi rohovkou a oční čočkou. Zmenšení průměru očního otvoru chrání oko před poškozením při nadměrném osvětlení. Za rohovkou je pružná oční čočka, jíž se obraz zaostřuje.

2.2.1 Oční čočka

Oční čočka je spojka s proměnnou optickou mohutností (kolem 10 D) [4]. Skládá se z vláken rozložených v šesti směrech. Čočka umožňuje ostré vidění blízkých i vzdálených předmětů. Její schopnost přizpůsobovat ohniskovou vzdálenost se nazývá akomodace a umožňují ji zvláštní svaly. Rozsah vzdáleností, na které může oko akomodovat, je určen dvěma body. Největší vzdálenost představuje vzdálený bod oka a nejmenší vzdálenost, při níž ještě oko zobrazí předmět ostře, je blízký bod oka. U normálního oka je vzdálený bod v nekonečně velké vzdálenosti. Blízký bod oka může být i 15 cm a mění se s věkem. Vidění na tak malou vzdálenost je spojeno se značnou námahou a oko se brzy unaví. Nejmenší vzdálenost, v níž ještě člověk zřetelně vidí, je u dětí asi 100 mm a u dospělých přibližně 200 mm od oka. Tzv. správná vzdálenost pro čtení, psaní a pozorování drobných předmětů je asi 250 mm. Soustavným pozorováním předmětů z bližší vzdálenosti se oko může poškodit.

2.2.2 Sítnice

Oko vytváří obraz předmětů, které se nacházejí v různých vzdálenostech před okem, ve stejné vzdálenosti uvnitř oka, na citlivé sítnici. Obraz se tu vytváří zmenšený, převrácený a skutečný. Sítnici tvoří přibližně 130 milionů buněk. Jsou uspořádány do 3 vrstev.

2.2.3 Vnímání barev

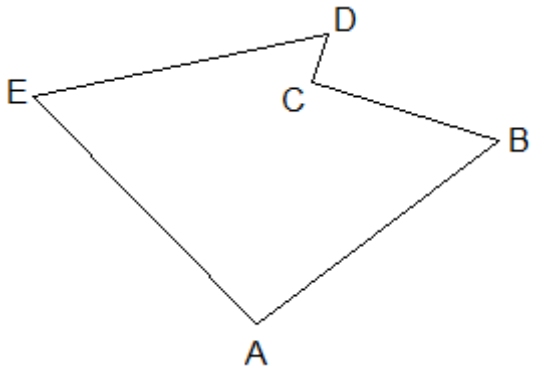
Buňky citlivé na světlo jsou uloženy až v nejhlubší vrstvě – jsou to tyčinky a čípky. Barevně vidíme prostřednictvím čípků. V šeru vidíme černobíle pomocí tyčinek, které jsou značně citlivější než čípky. Čípků je v oku 6,5 miliónů, tyčinek 125 miliónů. V sítnici nejsou rozloženy rovnoměrně. Ve střední části sítnice je více čípků než tyčinek - čípků radiálním směrem ubývá, takže na okraji převládají tyčinky. Žlutá skvrna obsahuje jen čípky, které umožňují barevné vidění. Čípky mohou být v plné činnosti při jasech větších než 10 cd * m^{-2}[4], tj. při fotopickém vidění (čípkovém). Při jasech menších než 10-3 cd * m^{-2} zůstávají v činnosti jen tyčinky – vidění skotopické (tyčinkové) v šeru. Při skotopickém vidění jsme barvoslepí. Přechodným stavem je vidění mesopické, při kterém se v závislosti na intenzitě osvětlení uplatní tyčinky i čípky. Je však namáhavé.

3 Detekce polygonů v obraze

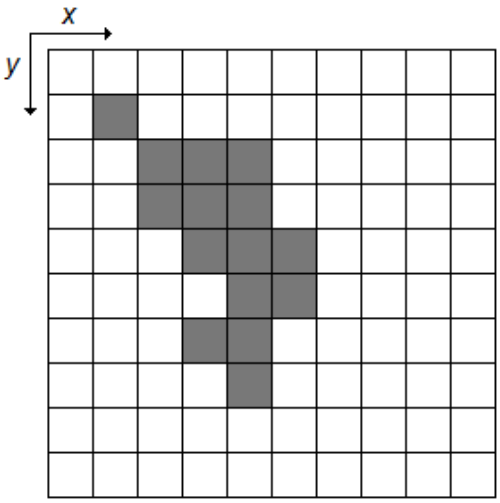
V této kapitole jsou stručně popsány definice polygonu a vybraných grafových algoritmů. Dále je zde vysvětlen postup, pomocí kterého program dokáže zjistit, které pixely v binárním obrázku tvoří jednotlivé polygony.

3.1 Definice polygonu

Polygon v planimetrii je definován jako: část roviny vymezená úsečkami, které spojují určitý počet bodů (nejméně tři), z nichž žádné tři sousední neleží na jedné přímce [1]. Počítačová grafika pracuje především s rastrovanými obrázky, jejichž základní jednotkou je pixel (nejmenší část obrázku, která je dále nedělitelná). Polygon si tak můžeme představit jako soubor všech pixelů stejné barvy, které spolu vzájemně sousedí a tvoří tak souvislou rovinu.



Obrázek č. 2: Polygon ABCDE v planimetrii



Obrázek č. 3: Polygon zaznamenaný v rastrové mapě 10x10 pixelů – bílé čtverce jsou pixely pozadí, šedivé čtverce jsou pixely polygonu

3.2 Výběr vhodného vyhledávacího algoritmu

Máme definovaný polygon, pro který teď potřebujeme najít vhodný vyhledávací algoritmus. Pro tyto účely si musíme představit obrázek jako graf, ve kterém budeme vyhledávat jeho souvislé komponenty. Každý pixel v obrázku bude reprezentován právě jedním uzlem, který bude mít následující atributy: *barva, souřadnice x, souřadnice y*. Po úvaze jsem došel k závěru, že vazba mezi uzlem *A* a *B*, neboli cesta z uzlu *A* do *B* a naopak, může vzniknout právě tehdy, když platí všechny tyto podmínky:

- $A_{RGBcolor} = B_{RGBcolor}$
- $A_x = B_{x,x\pm1}$
- $A_y = B_{y,y\pm1}$

Pakliže máme rastrovaný obrázek převeden na graf, můžeme vybrat vhodný algoritmus na prohledávání grafu. Potřebujeme, aby algoritmus prošel celou komponentou grafu co nejrychleji a testoval pouze její souvislost. Jako nejvhodnější adepti se proto jeví následující algoritmy:

- prohledávání grafu do hloubky *(Depth-first search)*
- prohledávání grafu do šířky *(Breadth-first search)*

3.3 Prohledávání grafu do hloubky

Algoritmus je úplný (vždy najde řešení, tzn. určitý cílový vrchol, pokud existuje), ale není optimální (pokud graf není strom, nemusí najít nejkratší možnou cestu k cíli). Dokáže projít pouze souvislé komponenty. Algoritmus prochází grafem tak, že nejdříve nastaví atribut *„stav“* všech uzlů na *„fresh“*. Poté se snaží projít co nejhlouběji do nitra grafu přes nenavštívené uzly (takové, které mají atribut *„stav“* roven *„fresh“*). Jakmile projde přes dosud nenavštívený uzel, nastaví jeho atribut *„stav“* na *„open“*. Pokud narazí na vrchol, z nějž už nelze dále pokračovat (nemá žádné následníky nebo byli všichni navštíveni), nastaví jeho atribut *„stav“* na hodnotu *„closed“* a vrací se zpět tzv. *backtrackingem*. Asymptotická složitost algoritmu: $O(|V| + |E|)$, kde V je počet vrcholů a E počet hran daného grafu [2].

3.4 Prohledávání grafu do šířky

Tento algoritmus má mnoho společných atributů s výše zmiňovaným algoritmem. Je úplný a dokáže prohledat pouze souvislé komponenty grafu. Stejným způsobem si označuje také atribut *„stav“* každého uzlu. Zásadní rozdíl oproti předešlému algoritmu je ve způsobu procházení grafem. Princip procházení je takový, že se snaží projít celý graf co nejvíce do šířky – tzn., jako první projde všechny potomky kořene, poté potomky těchto potomků. Výsledkem tohoto algoritmu je graf typu *„Tree“*. Asymptotická složitost algoritmu: $O(|V| + |E|)$, kde V je počet vrcholů a E počet hran daného grafu [3].

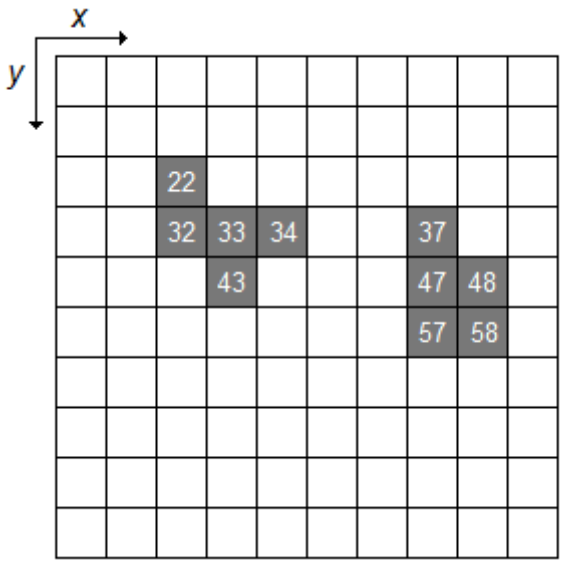
3.5 Návrh datových struktur vyhledávacího algoritmu

Oba výše zmíněné algoritmy mají totožnou asymptotickou složitost, proto ve výsledném programu nehraje zásadní roli rozhodnutí, který z těchto dvou algoritmů použijeme. Pro svůj program jsem si vybral algoritmus: *„Prohledávání grafu do šířky“*. Implementace bude vyžadovat tyto datové struktury:

- N_n - uzel (ponese atributy: *x*, *y*)
- *Data* - pole uzlů, které chceme analyzovat
- *Fifo* - fronta (paměť pro *BFS* algoritmus)
- *Polygon* – dvojrozměrné pole uzlů, ve kterém jednotlivé řádky pole uzlů přísluší jednomu polygonu

3.6 Návrh postupu pro vyhledávání polygonů pomocí BFS

Pokusím se zde zformulovat teoretický návrh postupu, jak najít jednotlivé polygony (souvislé komponenty grafu) v binárním rastrovaném obrázku. Jako příklad zdrojového obrázku, který může být podroben analýze pomocí BFS, je zde uveden obrázek č. 4, z něhož vyplývá, že bílé pixely (bílé čtverce) tvoří plochu pozadí a šedé čtverce tvoří pixely jednotlivých polygonů. Každý pixel má své jedinečné číslo (čísluje se od nuly a počátek je v levém horním rohu).



Obrázek č. 4: Dva polygony ve zdrojovém binárním obrázku s rastrem 10x10 pixelů

Nejdříve proběhne inicializace datové struktury typu *„N“* (uzel) a to tak, že každý pixel, který potřebujeme analyzovat (čili takové uzly, které mají jinou barvu, než je barva pozadí), převedeme na uzel *„N_n“*, přičemž atribut *„x“* bude roven vzdálenosti pixelu od

počátku osy o_x a atribut *„y“* bude roven vzdálenosti pixelu od počátku osy o_y. Poté proběhne inicializace struktury *„Data“*, do které se uloží všechny uzly *„N_n“*, které budeme podrobovat zkoumání. Dále se inicializují struktury *„Fifo“* a *„Polygon“*, které jsou na začátku vyhledávacího procesu prázdné. Průběh vyhledávacího procesu je popsán níže.

Příklad inicializace datových struktur podle obrázku č. 4:

- N_{22}: $x = 2, y = 2$
- ...
- N_{58}: $x = 5, y = 8$
- $Data = \{N_{22}, N_{32}, N_{33}, N_{34}, N_{37}, N_{43}, N_{47}, N_{48}, N_{57}, N_{58}\}$

3.7 Minispecifikace pro BFS

Nyní popíšu minispecifikaci pro BFS algoritmus, která je předlohou pro implementaci v konkrétním programovacím kódu. V průběhu minispecifikace dochází k „vyjmutí“ prvků ze struktury „*Data“*, které nemusí být vždy na první pozici v tomto poli. Pokud k takovému odebrání dojde, ostatní prvky, nalézající se za vyjmutým prvkem, se posunou v poli o jednu pozici doleva (směrem k počátku pole) a velikost pole se zmenší o jedna. Výsledkem této minispecifikace je naplněná datová struktura *„Polygon“* typu dvojrozměrné pole, v němž každý řádek obsahuje uzly N_n, které přísluší právě jednomu polygonu.

Minispecifikace:

1. Jestliže struktura ***Data*** není prázdná:
    - Vyjmi první prvek z ***Data*** a vlož ho do ***Fifo*** (všechny prvky v ***Data*** se posunou o jednu pozici do leva)

    jinak:

    - Pokračuj na úkol č. 5

2. Prohledej okolí (vzdálenost ± 1 od zkoumaného uzlu) prvního prvku ve ***Fifo***. Každý nalezený uzel porovnej s uzly v ***Data***. Jestliže ***Data*** nalezený uzel obsahují:
    - Vyjmi tento uzel z ***Data*** a vlož je do ***Fifo***

    jinak:

    - Zahoď tento nalezený uzel

3. Vyjmi první prvek z ***Fifo*** a vlož ho na řádek do ***Polygon***
4. Jestliže struktura Fifo není prázdná:
    - Pokračuj na úkol č. 2

jinak:

- Jestliže struktura ***Data*** není prázdná, vytvoř ve struktuře ***Polygon*** nový řádek a do něho ukládej nové prvky
- Pokračuj na úkol č. 1

5. Ukonči vyhledávací proces

V kapitole *6.1.1 Třída PolygonFinder* je popsáno praktické ověření výše zmíněných teoretických předpokladů.

4 Existující řešení

Před započetím této práce jsem se snažil dostat již k realizovaným komplexním systémům, které by umožňovali uživateli ovládat PC pouze za pomocí očí. Zde jsem narazil na nemožnost získat jakékoli podrobnější informace. Je to zapříčiněno několika faktory. Jedním z nich je, že výzkum v tomto odvětví se často zabývá jen dílčí problematikou oproti návrhu komplexního systému. Dalším faktorem je, že toto odvětví se stalo atraktivním až s příchodem miniaturních výkonných kamer a rychlých osobních počítačů, což umožnilo zpřístupnit tuto technologii běžným uživatelům za relativně přijatelnou cenou. Hlavní faktor, způsobující nedostatek podrobných informací, je takový, že po dokončení výzkumu se takovýto systém uvede na trh jako komerční produkt nebo jako zdravotnické zařízení. To znamená, že autoři si posléze svůj produkt chrání, navíc v případě zdravotnického zařízení se musí podle zákona zaplatit nákladná a zdlouhavá certifikace.

Jedním existujícím zástupcem v tomto oboru je systém, který byl vyvinut v laboratořích ČVUT v České republice. Jmenuje se I4Control® a byl schválen a uveden na trh v září 2008 [7]. Jeho cena se pohybuje na hranici 40 000,- Kč [6].

3.1 Systém I4Control®

Systém je založen na VOG metodě, která zaznamenává oční pohyby prostřednictvím kamery.

Obrázek č. 5: Náhlavní souprava systému I4Control®

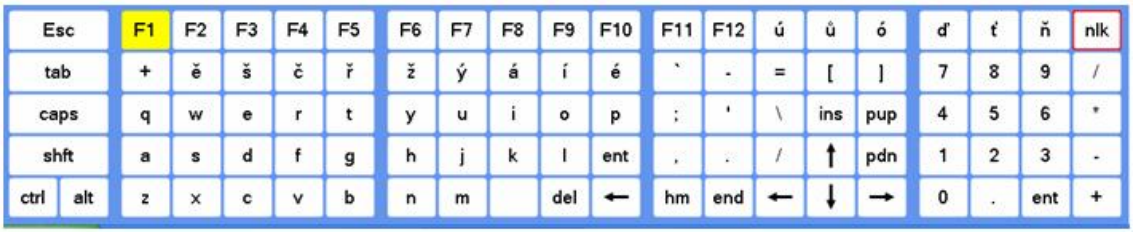
Kamera umístěná na brýlové obrubě snímá z bezprostřední blízkosti uživatelovo oko a jeho pohyby. Takto získaný videosignál se přenáší do řídicího modulu, ze kterého jsou data vedena prostřednictvím USB a videokabelu přímo do osobního počítače, kde jsou následně zpracována.

3.1.1 Ovládání kurzoru myši

Ovládání pohybu počítačového kurzoru probíhá v několika možných variantách. Žádná ovšem nenabízí způsob ovládání přímým pohledem, tj. přemístění počítačového kurzoru přímo tam, kam se uživatel dívá. Základní možností je přímé plynulé řízení počítačového kurzoru v inkrementálním režimu. Uživatel tedy ovládá přímo plynule pohyb kurzoru na monitoru výchylkami z klidové zóny (pohled přímo). Poloha oka nemá tedy přímo vliv na polohu kurzoru na monitoru, ale bude určovat pouze směr a délku jeho pohybu. Systém vyhodnotí aktuální polohu oka a podle jeho pozice vně/mimo klidové zóny buď provede či neprovede příslušné akce. Pokud je detekovaná zornice mimo klidovou zónu, systém neustále vysílá příkazy pro pohyb kurzoru do příslušného směru (podle aktuální polohy oka v souřadném systému kamery či podle nastavení v příslušném dialogu), dokud se oko do klidové zóny nevrátí. K přerušení pohybu kurzoru dochází tedy buď návratem oka do klidové zóny nebo pokusem o kliknutí. Systém pomocí doprovodné aplikace umožňuje nahrazovat všechny funkce běžné počítačové myši. Zavřením oka na dobu jedné sekundy je provedeno kliknutí a zavření oka na dvě sekundy vyvolá menu doprovodné aplikace, které nabízí rozšiřující funkce.

3.1.2 Skokovité ovládání kurzoru myši

Druhou variantou je skokovité (diskrétní) řízení počítačového kurzoru v inkrementálním režimu. Pohyb kurzoru je ovládán stejným způsobem, jako v předchozím případě. Jediným rozdílem je jiný typ pohybu kurzoru. Zatímco v prvním případě se kurzor pohybuje plynule, ve druhém skokovitě skáče. Tento způsob pohybu je možné s výhodou využít u specielně upravených aplikací, kdy je pro uživatele výhodnější skokovité ovládání, např. při psaní na programové klávesnici (kurzor volí přímo celá tlačítka).



Obrázek č. 6: Softwarová klávesnice systému I4Control®, po které se kurzor myši pohybuje skokovitě

5 Návrh systému EyeControl

V této kapitole jsou popsány všechny nezbytné návrhy pro realizaci systému na ovládání PC pohybem očí, který jsem pojmenoval pracovním názvem EyeControl.

3.1 Volba kamery

Nezbytnou součástí systému EyeControl je i zařízení pro zachytávání snímků okolí. Jelikož chceme snímat obraz oka z nejmenší vzdálenosti, je rozhodujícím faktorem při výběru kamery tyto parametry:

- ohnisková vzdálenost snímací čočky
- FPS (počet snímků za sekundu)
- rozlišení snímku
- citlivost snímacího senzoru na světlo
- celková velikost a hmotnost kamery

3.1.1 Volba parametrů kamery

Ohnisková vzdálenost snímací čočky by měla být co nejnižší, aby kamera dokázala zaostřit lidské oko z co nejmenší vzdálenosti. U FPS se nevyžadují nadstandardní hodnoty, jelikož běžný standart se dnes pohybuje v rozmezí 25 – 30 FPS u běžných kamer a to je pro systém EyeControl velmi dostačující. Velice problematická je ovšem velikost rozlišení pořízených snímku na výstupu kamery. Čím větší, tím lepší je obraz a mnohem lépe se na něm dají identifikovat snímané objekty. To má však za následek, že se několikanásobně zvýší datový tok z kamery, který neúměrně zatíží procesor a zpomalí tak FPS kamery a chod celého PC. Zde se tedy jako optimální rozlišení jeví hodnota na úrovní 640 x 480 pixelů na snímek. Dalším podstatným faktorem je schopnost kamery distribuovat kvalitní obraz i za zhoršených světelných podmínek. Zde všeobecně platí, že běžné modely, pohybující se na poli levnějších produktů, mívají citlivost na světlo velmi špatnou. To se někteří výrobci snaží kompenzovat přidáním diod, produkující infračervené světlo a filtru, který je umístěn před čočkou kamery a je na infračervené světlo citlivý. To ovšem zvyšuje velikost, hmotnost, energetickou náročnost a hlavně cenu kamery. Posledními faktory je hmotnost a velikost snímacího zařízení. Jelikož potřebujeme snímat oko z bezprostřední vzdálenosti, musí být kamera dostatečně lehká, aby mohla být přichycena ke hlavě uživatele, a musí mít malé rozměry, aby zabírala co nejmenší zorné pole uživatele. Modely, které nejlépe splňují poslední dva parametry, spadají do kategorie *„web kamery“*.

3.1.2 Výběr kamery

Na základě zhodnocení výše uvedených parametrů, jsem se rozhodl pro systém EyeControl vybrat webovou kameru *Microsoft Lifecam NX-6000*. Tato kamera je kompromisem mezi vybranými parametry popsanými v předešlé stati. Hlavní výhodou tohoto modelu je jeho velikost a hmotnost. Zde jsou dostupné parametry kamery:

- typ obrazového snímače: CMOS senzor
- maximální rozlišení videa: 1600 x 1190 pixelů
- objektiv: širokoúhlý
- ostření: automatické
- FPS: až 30
- rozměry (V x Š x H): 25,2 x 68,4 x 32 mm
- hmotnost: 54 g

Obrázek č. 7: Webová kamera Microsoft Lifecam NX-6000 pro systém EyeControl

3.2 Vývoj mechanismu na uchopení kamery

Systém EyeControl vyžaduje, aby kamera mohla snímat oko z bezprostřední vzdálenosti. To vyžaduje kameru pomocí uchopovacího mechanismu přichytit k lidské hlavě. Jako základ takového mechanismu mi posloužila nastavitelná čelenka z cyklistické přilby, která dokáže měnit svůj obvod pomocí kolečka umístěného v její zadní části. K této čelence je pomocí kovových svorek přichycen jeden konce měděného drátu s ochranou plastovou izolací o průměru 4 mm. Druhý konec je oblepen lepicí páskou tak, aby byl aspoň 1,5 centimetrů široký a je na něho přichycena kamera pomocí svého přichycovacího systému. Tento drát je dostatečné pevný, aby udržel kameru, a zároveň umožňuje pomocí ohýbání měnit vzdálenost kamery od oka.

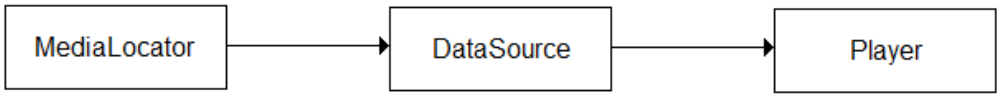
Obrázek č. 8: Mechanismus na uchopení kamery pro systém EyeControl

3.3 Analýza a výběr softwarových prostředků na realizaci systému EyeControl

Rozhodl jsem se realizovat systém EyeControl v objektovém, platformově nezávislém programovacím jazyce Java (StandartEdition v6.0). Jako vývojové prostředí jsem zvolil studio Eclipse v3.4 Ganymede. Pro práci s kamerou se v Jave používají knihovny technologie JavaMediaFramework (v2.0). Níže je popsána stručná charakteristika technologie JMF 2.0.

3.3.1 Způsob navazování spojení s kamerou

Práce s jakýmkoliv obrazovým zařízením, které podporuje multimediální framework *VideoForWindows*, je v JMF velmi jednoduchá. Postup je následující (podle obrázku č. 9).

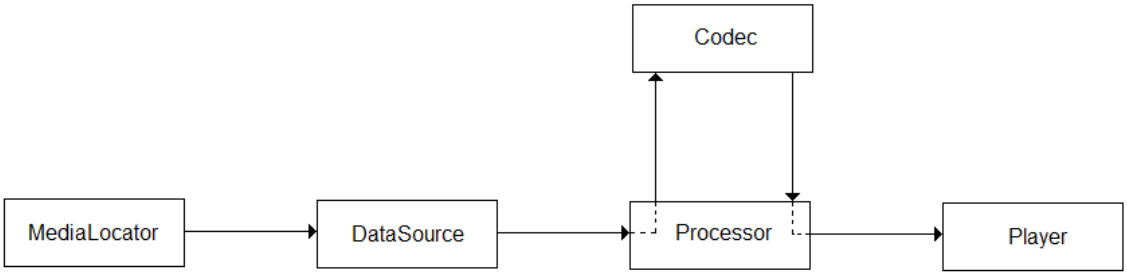


Obrázek č. 9: Jednoduché schéma získání obrazového výstupu z video kamery v JMF

Nejdříve se vytvoří instance třídy *MediaLocator* s parametrem URL: *„vfw://“* (VideoForWindows). Tato třída poté slouží jako parametr nové instance třídy *DataSource*, která je taktéž parametrem nové instance *Player*. Z třídy *Player* získáme obrazový výstup zavoláním její metody *getVisualComponent()*. Tímto máme k dispozici obrazový výstup z kamery, který můžeme libovolně umístit do *GUI* programu.

3.3.2 *Získání zdrojových dat obrázku z kamery*

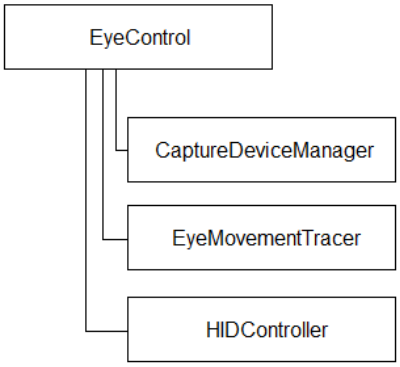
Pro systém EyeControl je však podstatné mít přístup přímo ke zdrojovým datům obrázku pořízeného kamerou. K tomu slouží *Processor* a *Codec*. Získání obrazového výstupu proběhne stejným způsobem, jako v předešlé stati, s tím rozdílem, že mezi objekty *DataSource* a *Player* se vloží objekt třídy *Processor*, který přesměruje datový tok na objekt *Codec* a poté poskytne svůj datový výstup (v podobě streamu) objektu *Player*. Třída *Codec* slouží ke zpracovávání dat z datového proudu záznamového zařízení, např. (de)kompresi, nebo k zakódování dat do jiného Audio/Video formátu. Tzn., že zde máme přístup k jednotlivým snímkům/vzorkům videa/audia. Systému EyeControl pouze stačí zmíněné data snímku zkopírovat, nezměněný originál dat poslat dále na výstup a kopii dat poslat ke zpracování.



Obrázek č. 10: Schéma získání zdrojových dat z kamery v JMF

3.4 Návrh struktury programu

Zde bych chtěl popsat návrh struktury a logického uspořádání tříd a jejich funkcí výsledného programu, který bude napsán v programovacím jazyce Java.



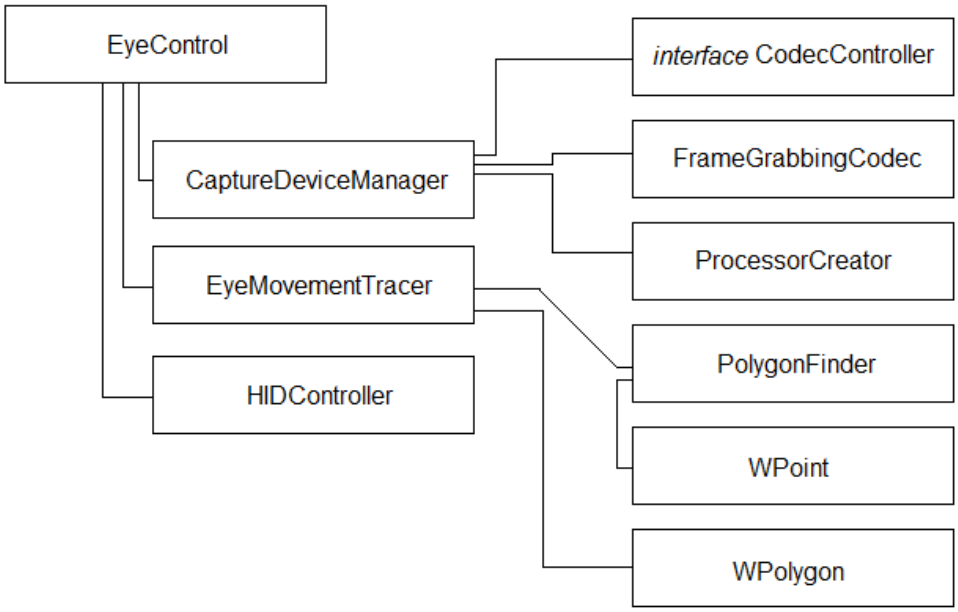
Obrázek č. 11: Návrh modulů systému EyeControl

Nejefektivnějším řešením je navrhnout strukturu programu tak, aby sestávala z několika základních modulů, které budou plnit dílčí úkoly systému *EyeControl* a které budou na sobě navzájem nezávislé (obrázek č. 11). Jejich běh bude řízen hlavním řídícím modulem, který bude také zajišťovat výměnu dat mezi ostatními moduly.

6 Realizace systému EyeControl

6.1 Třídy systému EyeControl

V této stati jsou popsány metody, atributy a funkce jednotlivých Java tříd systému *EyeControl*. Pro lepší představivost jsou na obrázku č. 9 třídy znázorněny jako závislé logické celky.



Obrázek č. 12: Uspořádání tříd v systému EyeControl

6.1.1 Třída PolygonFinder

V této kapitole jsou převedeny do praxe teoretické předpoklady z předešlé kapitoly. Je zde popsána třída programu *EyeControl*, která má za úkol v binárním rastrovém obrázku detekovat polygony a předávat je dál ke zpracování.

6.1.1.1 Atributy třídy PolygonFinder

Atributy třídy jsou téměř totožné s datovými strukturami, které byly popsány ve stati *„Návrh praktického použití vyhledávacího algoritmu“* v předešlé kapitole. Datová struktura *„N_n“* (uzel) je reprezentována objektem třídy *WPoint* (třída je popsána v kapitole 6.1.6). Další datová struktura *„Fifo“* je v této třídě reprezentována atributem s názvem *„queue“*, který je objektem třídy *Vector<WPoint>*. Datová struktura *„Polygon“* je zde reprezentována dvěma třídními atributy. První z nich nese název *„polygon“* a je objektem třídy *Vector<WPoint>*. Má za úkol nahrazovat roli „řádků“ v datové struktuře *„Polygon“*. Druhý z atributů nese název *„totalPolygons“,* je objektem třídy *Vector<WPolygon>* (třída je popsána v kapitole 6.1.7) a nese informace o všech nalezených polygonech. Dalším atributem, který je nutno zavést, je atribut *„baseColor“* nesoucí informaci, jakou barvu mají mít hledané pixely ve zdrojovém

obrázku. Atribut je primitivního datového typu *int*. Informace o barvě je v něm uložena jako součin desítkového vyjádření bitů tří barevných složek (*R*G*B*).

Největší změnu oproti původnímu návrhu prodělal reprezentant datové struktury *„Data“*. Ta byla původně zastoupena atributem *„data“*, který byl objektem třídy *Vector<WPoint>*. Před spuštěním vyhledávacího procesu se provedla inicializační metoda, která ze zdrojového obrázku načetla hodnoty (souřadnice *x*, *y*) všech pixelů požadované barvy a uložila je do atributu *„data“* v takovém pořadí, v jakém je našla (metoda postupovala po řádcích zleva doprava, začínala v levém horním rohu obrázku). Tímto byla provedena inicializace datových struktur přesně podle návrhu v předešlé kapitole a mohl se spustit samotný vyhledávací proces pomocí BFS algoritmu.

Testování v praxi ale přineslo katastrofální výsledky. Při použití zdrojového obrázku s rastrem 300x300 pixelů, který obsahoval jeden polygon zabírající přibližně 25% celkové plochy obrázku, dosahovaly časy, potřebné k dokončení celého vyhledávacího procesu nad jedním obrázkem, rozmezí mezi 20 – 25 sekundami (testováno na běžném PC s dvoujádrovým procesorem o celkovém taktu 2,1 GHz a operační pamětí 4 GB). Takový výsledný čas je pro praktické využití algoritmu zcela nepřijatelný.

Na základě výsledků testování jsem vyvodil následující závěry. Postup při inicializaci datové struktury *„Data“* (reprezentována třídním atributem *„data“*) je zcela nevhodný, jelikož setříděnou posloupnost prvků (všechny pixely jsou v obrázku setříděny vzestupně podle atributů *x*, *y*) převede na posloupnost nesetříděnou (ve výsledném poli jsou pouze pixely, které chceme testovat, ale hodnota atributu *x* nebo *y* neodpovídá indexu prvku v daném poli). Toto zjištění se projevilo v druhém kroku minispecifikace (kapitola 3.7) navržené pro BFS algoritmus. V něm se uvádí, že algoritmus má ověřit, jestli pro první prvek v datové struktuře *„Fifo“* existuje takový prvek, který by měl na rastrové mapě obrázku vzdálenost rovnu jedna od původního prvku, čili by takové dva prvky spolu těsně sousedily. Tento druhý krok minispecifikace by se mohl podrobněji rozepsat takto:

Podrobnější rozepsání 2. kroku minispecifikace pro BFS:

a) Prvek $\boldsymbol{N}$ = první prvek z ***Fifo***
b) Vytvoř 8 dočasných prvků $\boldsymbol{N}_{temp} = \{N_{x-1,y-1}, N_{x,y-1}, N_{x+1,y-1}, \ldots, N_{x+1,y+1}\}$
c) Pro každý prvek $\boldsymbol{N}_{temp}$ proveď:
    I. Otestuj, zdali se v poli ***Data*** nalézá právě takový prvek, který má stejné hodnoty atributů *x*, *y*, jako $\boldsymbol{N}_{temp}$

    jestliže takový prvek v poli ***Data*** existuje:

    II. Vyjmi tento prvek z pole ***Data*** a vlož ho nakonec pole ***Fifo***
d) Vyjmi první prvek z pole ***Fifo*** a vlož ho na řádek do pole ***Polygon***
e) Jestliže pole ***Fifo*** není prázdné, vrať se na krok a), jinak pokračuj dále

Z výše uvedeného postupu je nyní zcela jasné, která část celé minispecifikace je nejnáročnější na procesorový čas. Je to část *„2.-c)-I.“*, jelikož nutí algoritmus osmkrát pro každý prvek v poli *„Fifo“* projít polem *„Data“*. Nejkritičtější situace nastává, když algoritmus testuje prvky, které se nalézají na hranách polygonu, protože v tomto případě vytváří mnoho dočasných prvků, které se nachází vně polygonu v těsné blízkosti jeho hran. To má za následek, že algoritmus v této fázi musí polem *„Data“* projít od počátku, až do konce, aby se přesvědčil, že takovýto prvek neexistuje a že bylo dosáhnuto hrany polygonu.

Tyto poznatky mě donutili transformovat pole *„Data“* na takovou datovou strukturu, která by zachovávala setříděnost prvků v posloupnosti a umožnila tak jejich velmi rychlé vyhledávání. Nejjednodušším řešením, které se nabízí, je proto nahradit pole prvků za samotný zdrojový obrázek. Algoritmus nyní bude při vyhledávání prvků postupovat tak, že po vytvoření dočasného prvku N_{temp} se zaměří podle jeho atributů *x, y* na konkrétní pixel ve zdrojovém obrázku a otestuje jeho RGB hodnotu. Pokud barva neodpovídá barvě, kterou má nést hledaný polygon, algoritmus bude tento pixel považovat za pixel patřící pozadí obrázku a smaže tak dočasný prvek N_{temp}. V opačném případě bude prvek N_{temp} přidán do pole *„Fifo“* a příslušný pixel bude označen jako „objevený“ takovým způsobem, že jeho hodnota atributu RGB bude snížena např. o jeden bit.

Na základě všech těchto poznatků a výsledků testování jsem vybral jako vhodného reprezentanta datové struktury *„Data“* objekt třídy *BufferedImage* (balík: *java.awt.image*) a pojmenoval ho jako atribut *„sourceImg“*.

6.1.1.2 Nalezení kořenového pixelu

V kapitole *3.4 Prohledávání grafu do šířky* je uvedeno, že výsledkem BFS algoritmu je neorientovaný graf typu „strom“. Aby mohl BFS algoritmus otestovat souvislost komponenty, musíme do atributu *„queue“* (datová struktura *„Fifo“*) vložit první prvek, který bude výchozím prvkem (kořenem) pro vyhledávací proces algoritmu. O nalezení kořenového pixelu se v třídě *„PolygonFinder“* stará metoda *„findRootPixel()“*, která funguje na následujícím, velice jednoduchém principu.

Vyhledávací algoritmus si tuto metodu volá vždy, když je atribut *„queue“* prázdný, což může nastat jen ve dvou případech: algoritmus byl právě inicializován, a tak nebyl ještě vložen žádný prvek, nebo algoritmus již prošel všemi prvky, spadající pod daný kořen stromu (polygonu) a čeká na vložení nového kořene. Metoda po zavolání algoritmem začne procházet zdrojový obrázek od jeho počátku (levý horní pixel) a když objeví pixel, který má hodnotu atributu RGB stejnou jako atribut *„baseColor“*, vrátí algoritmu objekt třídy *„WPoint“*, který reprezentuje nalezený pixel. Algoritmus poté použije tento objekt jako výchozí (kořenový) prvek pro dohledání všech prvků, tvořící jeden polygon.

6.1.1.3 Prohledání okolí pixelu

Aby mohl vyhledávací algoritmus v obrázku nacházet souvislé komponenty, musí vědět, mezi kterými prvky (pixely) existuje vazba (cesta). K tomu slouží metoda *„scanNeighbourhood(WPoint node)“*, která prohledá pomocí metody *„includePoint“* (metoda je popsána v následující stati) těsné okolí prvku v jejím parametru. Postup prohledávání

je shodný s postupem v minispecifikaci pro BFS s tím rozdílem, že místo 8 dočasných prvků se vytvoří jeden prvek *„node“*, který v průběhu vyhledávání osmkrát změní hodnotu svých atributů *x, y*. Jestliže je nalezen pixel se stejnými atributy jako dočasný prvek *„node“*, je vytvořena kopie *„node“*, poté je vložena do pole atributu *„queue“* a nalezený pixel je označen jako *„OPEN“* (hodnota RGB pixelu je změněna o jeden bit).

6.1.1.4 Vyhledávání pixelů ve zdrojovém obrázku

O vyhledávání prvků ve zdrojovém obrázku se stará metoda *„includePoint“* s parametry *„WPoint p“*, *„boolean markAsOpen“* a návratovým typem *„boolean“*. Vyhledávání probíhá jednoduše pomocí atributů *x, y* objektu *WPoint*, který se metodě předá jako parametr. Pokud je pixel požadované barvy v obrázku nalezen a pokud je hodnota parametru *markAsOpen* rovna *„true“*, hodnota RGB barvy nalezeného pixelu je ve zdrojovém obrázku snížena o jeden bit. Metoda vrací *„true“*, pokud je pixel nalezen, jinak vrací *„false“*.

6.1.2 Třída CaptureDeviceManager

Tato třída má na starost obsluhu záznamového zařízení a extrahování jednotlivých snímků z videa.

6.1.2.1 Pořízení snímku z videa

Pořídit snímek z videa lze v této třídě dvěma způsoby. První způsob je takový, že k extrahování snímku dojde až ve chvíli, kdy o to požádá řídící třída (což je třída *EyeControl*) zavoláním veřejné metody *„getDataImage(): BufferedImage“*. Tím se spustí proces extrahování snímku pomocí kontroleru, který je standardní součástí JMF (třída *FrameGrabbingControl*). Druhý způsob se drží postupu, který je schematicky zobrazen a popsán ve stati *3.3.2-Získání zdrojových dat obrázku z kamery*. Slouží k němu metoda *„grabFrame(Buffer b): void“*, jež je implementací rozhraní *CodecController*. Zde se přistupuje opravdu ke každému snímku z kamery, což má za následek následující negativní jev – podstatné zvýšení zátěže procesoru.

6.1.2.2 Ovládání záznamového zařízení

Kamera se inicializuje zavoláním privátní metody *„initCaptureDevice(int captureMode, boolean sendMessageIfDone): void“*, která se vykoná v samostatném vlákně. Parametr *captureMode* je rozhodující v tom, jaký ze dvou způsobů extrahování snímku (předešlá stať) z videa se bude používat. Pokud je hodnota parametru *sendMessageIfDone* rovna *true*, po dokončení úspěšné inicializace zařízení se pošle zpráva řídícímu modulu (třída *EyeControl*). Provoz kamery se zastaví zavoláním veřejné metody *„stop(): void“*, která ukončí veškerou aktivitu záznamového zařízení.

6.1.3 Třída FrameGrabbingCodec

Tato třída implementuje rozhraní *javax.media.Codec* a slouží ke kopírování obrazových dat, která přicházejí z videokamery na rozhraní *javax.media.Processor*. Nejdůležitější funkci zde zastává metoda *„process(Buffer in, Buffer out)“*, která pracuje s parametry, jež jsou objekty třídy *Buffer*. V objektu *Buffer* je uložen aktuální snímek z kamery, jeho obrazový formát, časová

známka, způsob uložení dat (pole typu *byte, int,…*) a další atributy. Při zpracovávání dat v této metodě musíme dodržet základní postup:

1) zkopírovat obsah vstupního *bufferu* (povinné)
2) provést úpravu zkopírovaných dat (nepovinné)
3) naplnit výstupní *buffer* (upravenou) kopií vstupních dat

Pro systém *EyeControl* se uplatní krok č. 2, ve kterém nedojde k žádné úpravě dat, ale pouze si uchováme kopii vstupního *bufferu*, která se odešle k dalšímu zpracování na příslušný systémový modul.

6.1.4 Interface CodecController

Interface *CodecController* je založen na principu posluchače registrovaného na konkrétní událost. Je využíván třídou *CaptureDeviceManager*, která ho implementuje a registruje na objekty třídy *FrameGrabbingCodec*. Tento interface je používan tak, že reaguje na událost *„příchod nového snímku z videokamery do objektu Codec“*. Nový snímek je předán metodě *CodecController.grabFrame(Buffer buffer)* pomocí parametru *buffer*.

6.1.5 Třída ProcessorCreator

Třída slouží k realizaci rozhraní *Processor*, čímž dojde k inicializaci záznamového zařízení (např. videokamery).

6.1.6 Třída WPoint

JavaSE6 v balíku *java.awt* obsahuje třídu *Point,* která je takřka stejná s třídou *WPoint*. Rozdíl je v počtu atributů. Třída *WPoint* se používá k reprezentaci pixelů na rastrové mapě obrázku a byla navržena tak, aby splňovala požadavky třídy *PolygonFinder*, které jsou náročné na paměť PC. Proto třída *WPoint* je odlehčenou kopií třídy *java.awt.Point*. Nese pouze tři atributy: *int x, int y* a *boolean isBorderPoint*. Atribut *isBorderPoint* vyjadřuje, jestli pixel, který je reprezentován touto třídou, leží na okraji polygonu.

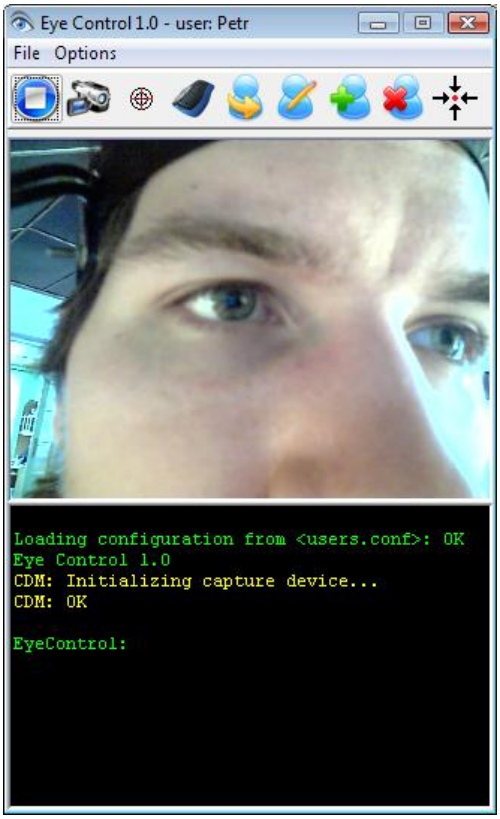
6.1.7 Třída WPolygon

Tato třída taktéž vychází z třídy *java.awt.Polygon* (ze stejných důvodů jako třída *WPoint*). Slouží pro ukládání polygonů, které jsou tvořeny objekty *WPoint*. Pro systém *EyeControl* je důležitý tzv. rámec polygonu, což je obdélník, jehož strany se dotýkají severního, jižního, východního a západního bodu polygonu. Tyto body dokáže třída *WPolygon* najít pomocí svých metod: *getTopUp(),getTopDown(),getTopLeft(),getTopRight()*.

6.1.8 Řídící třída EyeControl

Tato třída je hlavním řídícím prvkem. Zajišťuje chod a výměnu dat mezi ostatními moduly. Uživateli systému *EyeControl* umožňuje pomocí svého grafického rozhraní a konzole

spravovat a řídit chod aplikace. Mezi její funkce patří správa a podpora víceuživatelského rozhraní, kalibrace systému, poskytování náhledu videa ze záznamového zařízení a další.



Obrázek č. 13: GUI třídy EyeControl

6.1.9 Třída EyeMovementTracer

Tato třída je důležitým prvkem systému *EyeControl*, jelikož vyhodnocuje data z třídy *PolygonFinder* (které ji předtím poskytla) a odesílá je přes řídící modul do třídy *HIDController*.

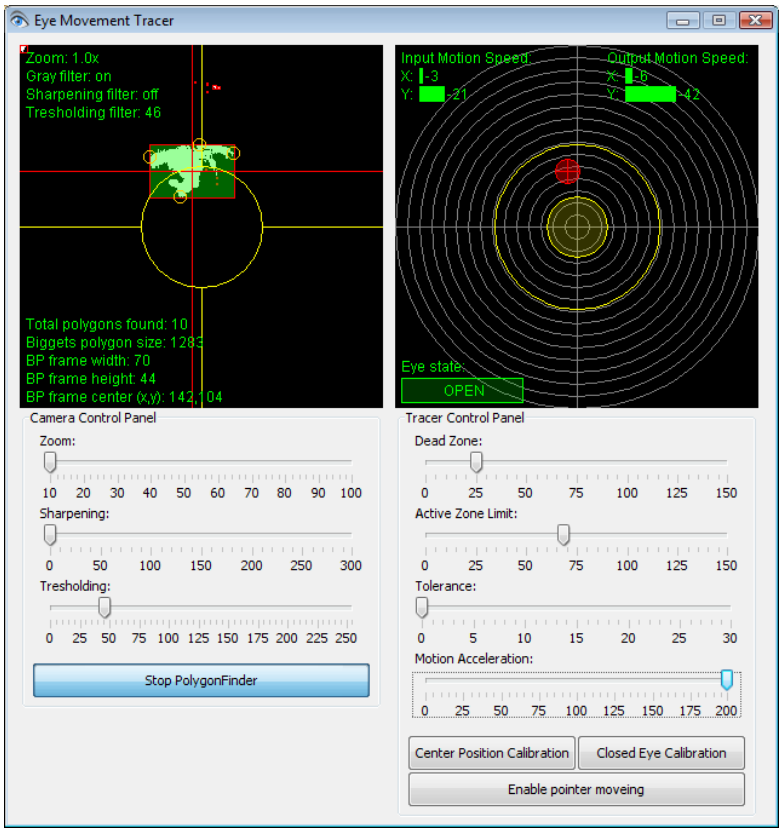
6.1.9.1 Zpracování obrazu z kamery

K zpracování obrazu používá tato třída metodu *„processImage(BufferedImage img, Graphics2D g): void“*, která postupně volá a provádí ve zdrojovém obrázku následující procedury:

- použití „šedého“ filtru (zdrojový obrázek má pouze barvy odstínů šedi)
- zoom (obrázek může být digitálně zvětšen pomocí bikubického škálování)
- ostření hran v obrázku
- prahování obrázku

Obsluha systému se pomocí různých kombinací těchto procedur a měněním hodnot jejich atributů snaží dosáhnout ideálního stavu, kdy je na výsledném snímku dobře čitelná poloha oka. Nejvýznamnější roli zde hraje tzv. prahování snímku (*tresholding*), které převede původní

obrázek na binární (pixely mají jen černou nebo bílou barvu). Po této proceduře může být snímek postoupen analýze v třídě *PolygonFinder*, která v něm nalezne všechny zobrazené polygony bílé barvy. Rozhodující je největší nalezený polygon, který bude považován za oko. Vypočte se osový střed jeho rámce, jehož poloha je přenesena na vztažnou rovinu, ze které se určí pohyb oka.



Obrázek č. 14: GUI třídy EyeMovementTracer

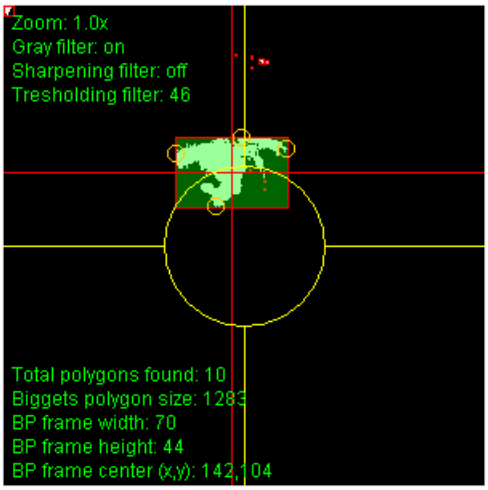
6.1.9.2 Analýza pohybu a stavu oka

Systém *EyeControl* vyhodnocuje vzdálenost zornice od „klidové“ polohy, při které oko sleduje objekt uprostřed svého zorného pole. Velikost této vzdálenosti se přepočte na rychlost (počet pixelů na osách *x*, *y* za jeden vyhodnocený snímek z kamery) pohybu kurzoru myši daným směrem. Tzn., že kurzor myši se nepřemístí do oblasti, na kterou je zaměřeno lidské oko, nýbrž se bude tímto směrem pohybovat, dokud se oko nevrátí do klidové polohy.

Simulace zmáčknutí tlačítka myši uživatelem je řešena pomocí rozpoznávání stavu oka. Pokud uživatel má po dobu 1,5 sekundy zavřené snímané oko, systém provede „zmáčknutí“ levého tlačítka myši. Pokud je oko zavřené 2,5 sekundy, provede se „zmáčknutí“ pravého tlačítka myši. Systém dokáže rozpoznávat stav oka pomocí kalibrace, při které se uloží do paměti systému proporce rámce polygonu (šířka, výška) a jeho střed ve chvíli, kdy má uživatel oko zavřené. Po kalibraci se v každém snímku z kamery porovnávají hodnoty největšího polygonu s hodnotami uloženými během kalibrace. Při tomto porovnávání se samozřejmě počítá s odchylkou, jejíž velikost si může uživatel sám nadefinovat.

6.1.9.3 Grafické zobrazení vstupních a výstupních dat uživateli

Součástí GUI třídy *EyeMovementTracer* jsou také grafické ukazatele, které zajišťují uživateli přehled o hodnotách vstupních a výstupních dat této třídy (obrázek č. 15).



Obrázek č. 15: Největší polygon (v červeném rámci) ze snímku z kamery, jak ho detekovala třída PolygonFinder

Vykreslování probíhá pomocí metod *„paintStatsForCamera(Graphics2D g): void“* a *„paintStatsForMovementTracer(Graphics2D g): void“*.

6.1.10 Třída HIDController

Tato třída má za úkol simulovat hardwarovou klávesnici a PC myš. V případě klávesnice to umožňuje uživateli pomocí svého grafického rozhraní (obrázek č. 16). Pohyb myší se provádí na základě polohových dat, které jsou doručovány z třídy *EyeMovementTracer*.

Obrázek č. 16: GUI třídy HIDController

6.2 Komunikace tříd s řídícím modulem

Ke komunikaci tří hlavních modulů (třídy *CaptureDeviceManager*, *EyeMovementTracer* a *HIDController*) s řídícím modulem (třída *EyeControl*) se využívá technologie *JavaBeans*, konkrétně třídy *PropertyChangeListener*. Řídící modul si zaregistruje na ostatní třídy posluchače událostí typu *PropertyChangeEvent*. Jména událostí musí dodržovat tyto podmínky. V první části jména je uveden zdroj, který událost vyvolal, oddělovací znak (středník) a jméno hodnoty, nebo zpráva. Např. událost o dokončení inicializace kamery v modulu *CaptureDeviceManager* by vypadala následovně: *„captureDeviceManager; captureDeviceIsReady“*. Řídící modul při dekódování zprávy postupuje jako deterministický automat pomocí třídy *StringParser*, kdy z jednoho řetězce udělá pole podle oddělovacího znaku.

6.3 Kalibrace systému

Aby systém správně fungoval, je nutno po jeho spuštění provést kalibraci. Při té se stanoví tzv. klidová poloha, kdy kurzor myši nebude na polohu oka reagovat pohybem. Střed této polohy je obklopen kruhovou oblastí, která se nazývá „mrtvá zóna“. Systém ji vypočte tak, že po krátký časový interval nahrává pozici oka v klidové poloze. Z nahraných dat použije největší odchylku, která bude zároveň udávat hranici mrtvé zóny. V dalším kroku systém vyzve uživatele, aby pohnul okem co nejvíce doprava. Tím se určí hranice aktivní zóny. Za touto linií systém na pohyb oka nebude reagovat. Posledním krokem kalibrace je uložení dat při zavřeném oku. Systém vyzve uživatele, aby zavřel oko a poté si zaznamená atributy největšího polygonu nalezeného ve zdrojovém obrázku z kamery. Po dokončení kalibrace je systém připraven k použití. V průběhu užívání systému může dojít ke změně hodnot klíčových atributů kalibrace, např. vlivem změny intenzity osvětlení. Uživatel tak může opětovat celou kalibraci, nebo jenom její dílčí části.

6.4 Víceuživatelské rozhraní

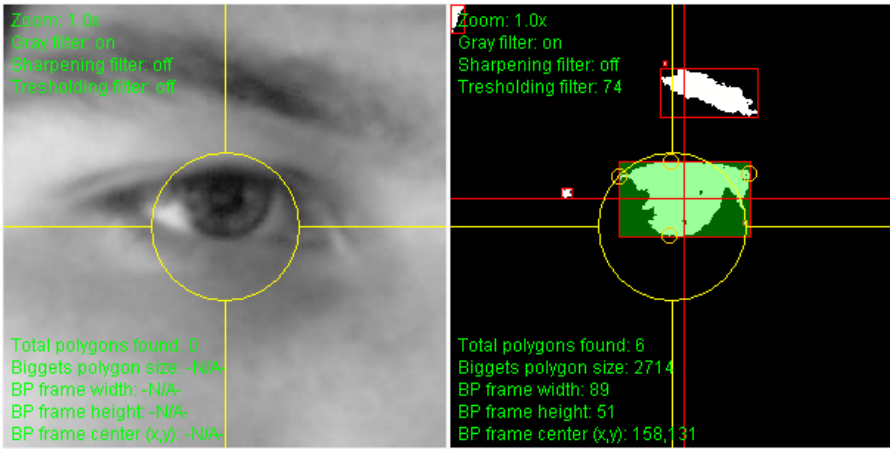
Systém umožňuje podporu více uživatelů. Slouží k tomu sada metod, které ukládají do konfiguračního souboru sadu všech hodnot, které může uživatel během používání systému měnit pomocí grafického rozhraní programu. Jde např. o hodnoty úrovně prahování, digitální přiblížení oka, mrtvé/aktivní zóny atd.

7 Testování systému

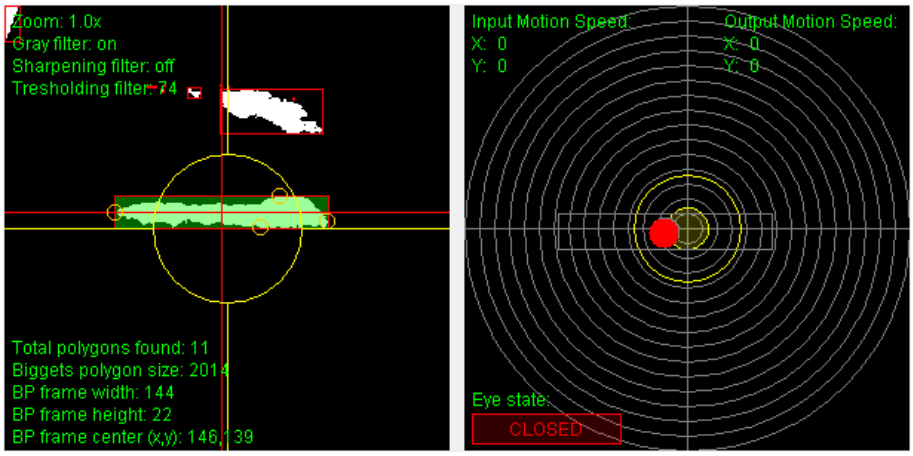
V této kapitole je popsáno testování systému *EyeControl* a jsou zde uvedeny poznatky, které byly z tohoto testování vyvozeny.

7.1 Rozpoznání oka ze snímku z kamery

Zásadním problémem v systému *EyeControl* je extrahovat tvar zornice z celého obrazu oka. Zornice má černou barvu, a tak je dobře patrná na bílém pozadí oka. Systém registruje černé objekty v obraze a převádí je na polygony. Zornice je tedy rozpoznána dobře, nicméně je doplněna o tmavé pixely, které tvoří řasy na očním víčku. Aby se tento problém eliminoval, musí se snížit citlivost prahování obrázku. To má ovšem za následek zhoršení obrazu zornice.



Obrázek č. 17: Obraz z kamery po použití metody prahování



Obrázek č. 18: Uživatel zavřel oko a systém tuto akci rozpoznal

Dalším ze způsobů, jak docílit kvalitního zaměření zornice v obrazu, je udržovat oko více otevřené, než je běžné, a osvětlit ho externím zdrojem světla.

7.2 Poznatky z testování

Během testování systému *EyeControl* jsem zjistil, že špatné zaměření zornice oka je způsobeno dvěma hlavními faktory.

Prvním z nich je, že vybraný model videokamery pořizuje sice snímky v dostatečně velkém rozlišení, nicméně nedokáže dobře zaostřit na blízké předměty. To má za následek, že jsou snímky neostré a zachycené objekty na nich mají rozmazané kontury, takže se špatně zaměřují.

Druhým faktorem je vysoká citlivost kamery na světelnou intenzitu. Toto je zásadní hardwarový nedostatek, který velmi zhoršuje schopnost systému zaměřit zornici lidského oka při snížené hladině osvětlení. Výše zmíněné faktory způsobují, že systém nedokáže plně využít celý rozsah pohybu oka. Kurzor nereaguje rovnoměrně na všechny strany a přestává reagovat úplně již v poměrně malé vzdálenosti od jeho mrtvé zóny. Toto ale lze kompenzovat tzv. systémovou akcelerací rychlosti pohybu.

8 Závěr

Cílem této práce bylo realizovat komplexní systém, který by pomocí snímání obrazu lidského oka videokamerou umožnil uživateli ovládat PC. To vše za použití běžných softwarových a hardwarových prostředků. Autor se snažil nejdříve problém analyzovat a poté zformulovat postupy, které by vedly k úspěšné realizaci takovéhoto systému.

Výsledkem implementace těchto návrhů je systém *EyeControl*, který splňuje výše zmíněné požadavky. Tento systém je funkční a připraven k použití uživatelům, pro které byl určen (lidé s postižením svalového ústrojí a se sníženou motorikou končetin). Nicméně trpí několika nedostatky, které jsou ale dány především povahou použitých hardwarových prostředků.

Díky rozdělení dílčích funkcí systému do jeho jednotlivých modulů je systém připraven pro snažší upgrade a případné zdokonalování, které by uživateli zlepšilo ovládání PC. Např. skokovité ovládání myši tak, jak je popsáno v kapitole 3.1.2. Lepší výsledky by také přineslo použití dokonalejší videokamery, která by měla nízkou ohniskovou vzdálenost čočky (makro-kamera), externí zdroj infračerveného světla a filtr citlivý na toto záření.

9 Seznam použité literatury

[1] *Wikipedia, Mnohoúhelník*, 10. 4. 2009, <http://www.wikipedia.org>

[2] *Wikipedia, Prohledávání do hloubky*, 16. 4. 2009, <http://www.wikipedia.org>

[3] *Wikipedia, Prohledávání do šířky*, 16. 4. 2009, <http://www.wikipedia.org>

[4] *Matematicko-fyzikální web*, 19. 4. 2009, <http://mfweb.wz.cz>

[5] *Kompendium oční optiky*, 20. 4. 2009, <http://www.zeiss.de>

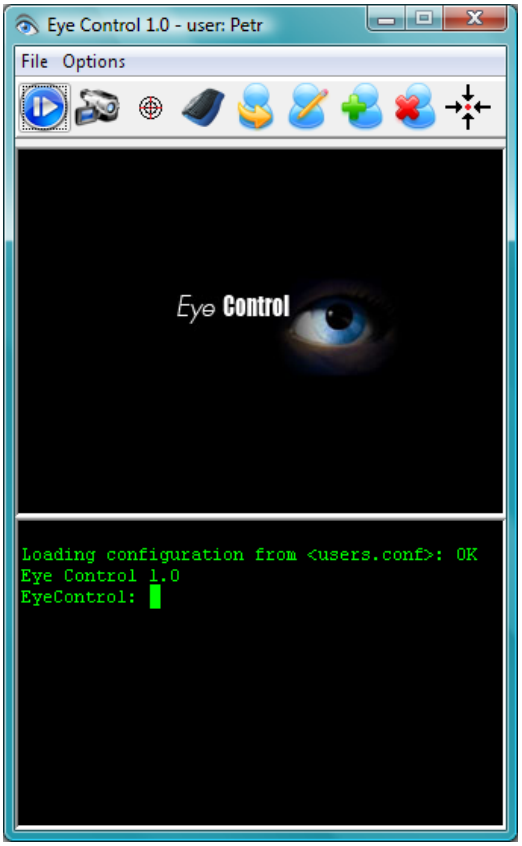
[6] TOMEK, P. Surfovat očima, *VTM Science*, 2009, vol. 63, no. 2, s. 38-39.

[7] Systém I4Control, 1. 5. 2009, <http://www.i4control.eu>

[8] Kotas, Oldřich, *Sledování oka a určování směru zájmu lidského pohledu* [diplomová práce], Ostrava 2006

A Uživatelská příručka systému EyeControl

Program EyeControl se inicializuje spuštěním souboru *„EyeControl.jar“*, archiv obsahuje všechny potřebné knihovny.

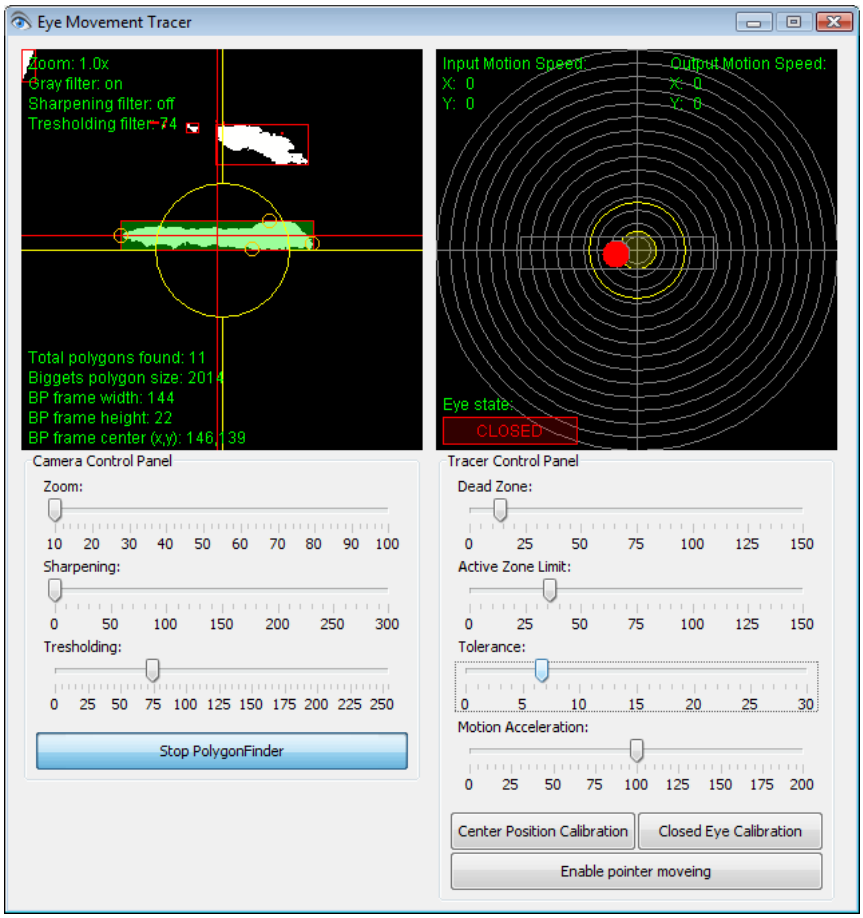


Obrázek č. 1: Hlavní okno Eye Control

První ikona z leva v horní části okna *Eye Control*(obrázek č. 1) aktivuje videokameru a umožní analyzovat snímky v okně *Eye Movement Tracer* (obrázek č. 2). Druhá ikona pouze inicializuje nahrávací zařízení. Třetí ikona zobrazí okno *Eye Movement Tracer.* Čtvrtá ikona zobrazí okno *Eye Control Keyboard* (obrázek č. 3). Pátá ikona umožní vybrat jiného uživatele systému. Šestá ikona uloží aktuální nastavení systému aktivnímu uživateli. Sedmá ikona umožní přidat nového uživatele. Osmá ikona odebere neaktivního uživatele. Devátá ikona spustí kalibraci systému.

Nyní bude následovat popis okna *Eye Movement Tracer* (obrázek č. 2). Jezdec *zoom* určuje hodnotu digitálního zvětšení vstupního obrázku. Jezdec s*harpening* zvýrazňuje hrany ve zdrojovém obrázku. Jezdec *tresholding* určuje hodnotu prahu vstupního obrázku. Tlačítko *start/stop PolygonFinder* spustí detekci polygonů ve vstupním obrázku. Podmínkou je hodnota prahu vyšší nebo rovna jedné. Jezdec *dead zone* určuje velikost oblasti, ve které bude kurzor

vztažné roviny neaktivní (nebude se brát v úvahu jeho pohyb). Jezdec *active zone limit* určuje hranici oblasti, uvnitř které bude probíhat výpočet pohybu kurzoru. Jezdec *tolerance* určuje velikost odchylky, která je přípustná pro rozpoznávání stavu oka (zavřené/otevřené). Jezdec *motion acceleration* vyjadřuje, o kolik procent se má zrychlit/zpomalit pohyb myši. Tlačítko *center position calibration* provede pouze kalibraci střední polohy kurzoru. Tlačítko *closed eye calibration* provede pouze kalibraci hodnot určující stav zavřeného oka. Tlačítko *enable pointer moveing* umožní aplikovat vypočítaný pohyb oka na pohyb myši. Před aktivací je velmi důležité mít správně systém zkalibrovaný. Hodnota *active zone* by neměla být větší, než je nezbytně nutné.



Obrázek č. 2: Okno Eye Movement Tracer

Nyní bude následovat popis okna *Eye Control Keyboard* (obrázek č. 3). Každé tlačítko v tomto okně reprezentuje příslušné tlačítko hardwarové klávesnice. Speciální funkci má tlačítko *settings*, které otevře dialogové okno, v němž jde pomocí jezdce měnit velikost celé softwarové klávesnice.

Obrázek č. 3: Okno Eye Control Keyboard

B Obsah CD přílohy

Soubory uložené na CD:

- text.pdf – text bakalářské práce
- EyeControl.jar – zkompilovaný program, zdrojové třídy, programátorská dokumentace